The Center for Information Systems
Security Studies and Research

Technical Report NPS-CS-05-003

A Least Privilege Model for Static Separation Kernels

Timothy E. Levin, Cynthia E Irvine, Thuy D. Nguyen

October 2004

Report Documentation Page

Form Approved
OMB No. 0704-0188

Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.

1. REPORT DATE **OCT 2004**	2. REPORT TYPE	3. DATES COVERED **00-00-2004 to 00-00-2004**
4. TITLE AND SUBTITLE **A Least Privilege Model for Static Separation Kernels**		5a. CONTRACT NUMBER
		5b. GRANT NUMBER
		5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER
		5e. TASK NUMBER
		5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Naval Postgraduate School,The Center for Information Systems Security Studies and Research,Monterey,CA,93943-5000**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S)
		11. SPONSOR/MONITOR'S REPORT NUMBER(S)
12. DISTRIBUTION/AVAILABILITY STATEMENT **Approved for public release; distribution unlimited**		
13. SUPPLEMENTARY NOTES		
14. ABSTRACT **see report**		
15. SUBJECT TERMS		

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT **Same as Report (SAR)**	18. NUMBER OF PAGES **21**	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**			

A Least Privilege Model for Static Separation Kernels

Timothy E. Levin, Cynthia E Irvine, Thuy D. Nguyen
Naval Postgraduate School
Monterey, CA

Abstract

We extend the separation kernel abstraction to represent the enforcement of the principle of least privilege. In addition to the inter-block flow control policy prescribed by the traditional separation kernel paradigm, we describe an orthogonal finer-grained flow control policy by extending the protection of elements to subjects and resources, as well as blocks, within a partitioned system. We show how least privilege applied to the actions of subjects and resources provides enhanced protection for secure systems, and how only "trusted subjects" may cause certain information flows between partitions. A high assurance separation kernel based on least privilege can provide all of the functionality and protection of the traditional separation kernel, combined with a high level of confidence that the effects of subjects' activities can be minimized to their intended scope.

1 Introduction

The Sisyphean purgatory of penetrate and patch to which users of commodity systems are currently subjected has lead to increasing recognition that platforms with assurance of penetration resistance and non-bypassability are required for certain critical functions. This need for high assurance calls for a layered system architecture where enforcement mechanisms of the most critical policies themselves depend upon layers of no less assurance. In the past, the solution offered was a general-purpose security kernel. For many high assurance systems currently being planned or developed, a general-purpose kernel provides more functionality than necessary. This has resulted in increased interest in the use of separation kernels to support real-time embedded systems and as the basis for virtual machine monitors (VMM). Many of these separation kernels are minimized to have both static policies and static allocation of resources, such as would be suitable for certain fixed-configuration or embedded environments.

The relevance of the separation kernel concept is demonstrated by the many current systems and projects that utilize them. A few U.S. systems where embedded separation kernels will be deployed include: the F-35 Joint Strike Fighter, F-22, Joint Tactical Radio System, C-17 Globemaster III, and Unmanned Combat Air Vehicle [27]. The JSF requires the evaluation and certification at EAL7 [27]. Separation kernels will also be used in communications security modernization efforts, where a single processor board will be used to manage cryptography for information at a variety of classification levels [6]. VMM implementations in development include VMware [10], the Xen project [3], the Denali project[44], and Microsoft's NGSCB project [5][11][29][12].

Despite the resurgence of interest in the separation kernel approach, the *principle of least privilege* [39] is often overlooked in the design of traditional separation kernels due to the belief that a separation kernel should only be concerned with resource isolation. Consequences of this omission include problems relating to all-or-nothing security and over-privileged programs. For systems that must protect highly sensitive or highly valuable resources, e.g., the JSF, formal verification of the ability of the system to enforce its security policy is required. Recent advances in the assurance requirements for high assurance systems[31] have included verification of the target system's conformance to the principle of least privilege. To provide vendors and integrators with tools to formally describe least privilege in separation kernels, a least privilege separation model is offered.

1.1 A *Least Privileged* Separation Kernel

In the context of a research project to build a high assurance separation kernel, we have extended the separation kernel abstraction so that the principle of least privilege can be examined at the model level and enforced by systems that conform to that model.

The traditional separation kernel paradigm describes a security policy in which activities in different *blocks* of a partitioned system are not visible to other blocks, except perhaps for certain specified flows allowed between blocks. (Here, "block" is defined in the traditional mathematical sense as a member of the non-intersecting set of elements that comprise the partition [34].) If information flow is described only at the block level, then everything in a block can flow to everything in another block. This would be contrary to the principle of least privilege required in high assurance systems. The least privilege separation model presented here builds on the traditional separation abstraction by extending the granularity of described elements to the subjects [25] and resources within the partition. An orthogonal flow control policy can then be expressed relative to subjects and resources. Thus, a high assurance separation kernel can provide all of the functionality and protection of the traditional separation kernel, combined with a high level of confidence that the effects of subjects' activities may be minimized to their intended scope.

In the sections that follow we will elaborate on the concept of separation kernels and the need for least privilege in such systems. In particular, the granularity of inter-block flows will be discussed and the requirement for the subject abstraction will be described. A formalization of the least privilege separation model is presented and several aspects of secure system design and verification are discussed with respect to the model. The last sections of the paper review related work, and summarize our results.

2 Concepts

2.1 The Separation Kernel

The term *separation kernel* was introduced in the early 1980s and has since become a term of art [37]. Rushby originally proposed, in the context of a distributed system, that a separation kernel creates “within a single shared machine, an environment which supports the various components of the system, and provides the communications channels between them, in such a way that individual components of the system cannot distinguish this shared environment from a physically distributed one.”

A separation kernel divides all resources under its control into blocks such that the actions of an active entity (i.e., a subject) in one block are isolated from (viz., cannot be detected by or communicated to) an active entity in another block, unless an explicit means for that communication has been established (e.g., via configuration data). Separation kernels are typically employed as the basic underlying component of many embedded applications, operating systems, or virtual machine monitors. For a multilevel secure system, each block typically represents a different classification level.

A separation kernel achieves isolation of subjects in different blocks by virtualization of shared resources: each block encompasses a resource set that appears to be entirely its own. To achieve this objective for resources that can only be utilized by one subject at a time, such as the CPU, the ideal separation kernel must ensure that the temporal usage patterns from different blocks are not apparent to each other. Other resources, such as memory, may be accessed by different blocks simultaneously, while preserving idealized isolation, if the separation kernel ensures, for example, that blocks are allocated different and non-interacting portions of the resource. Furthermore, kernel utilization of its own internal resources must also preserve the desired isolation properties.

2.1.1 Application of the Separation Concept

The conceptual model of separation is commonly used to create Virtual Machine Monitors (VMM), which provide efficient, isolated programming environments that replicate the real machine environment. These mimicked environments are referred to as virtual machines. [17][33]. However, a separation kernel differs from a VMM in that it is not required to exactly replicate the underlying hardware base to to its application. Preventing illegal information flow between isolated blocks is the primary common functionality between a separation kernel and a VMM, and is the reason that the separation kernel provides an ideal foundation upon which to build a secure VMM.

Goldberg defines a virtual machine (VM) as: "a hardware-software duplicate of a real existing computer system in which a statistically dominant subset of the virtual processor's instructions execute on the host processor in native mode" [17]. A VMM mediates between the virtual machine(s) and the real resources of the computer system. Benefits of VMMs include the ability of the VMM administrator to configure the execution environment of the VMs, support for the concurrent use of several different operating systems, isolation of the VMs, and flexibility for upgrades or modifications to the VM operating systems and applications. If both security and performance are requirements, then, to succeed, the partitioning performed by the VMM must ensure that the virtual machines are isolated from one another such that information flows between them are constrained, while providing support for a variety of operating systems and imposing minimal performance degradation.

Process isolation is another application of the resource separation concept. The separation mechanism to isolate execution domains (e.g., process address spaces) such that one domain cannot interfere with another domain has been implemented in both hardware (e.g., Intel IA-32 architecture [20]) and the security kernel [13]. Unlike a traditional security kernel that enforces a multilevel access control policy and performs the most trusted functions for a secure operating system, a separation kernel concerns itself only with the partitioning of resources and flow control.

When applied to embedded military systems, the separation kernel's resource partitioning is commonly used to enforce both avionics-related policies and security policies. As these safety-oriented avionics systems are used increasingly to process sensitive data, they will be scrutinized with respect to high-level standardized functional security and assurance requirements. Such criteria differ from the assessment standards for flight safety certification (e.g. RTCA/DO-178B Level A [36]) in that they articulate security's negative requirement that the absence of unspecified functionality be demonstrated: safety requirements provide no assurance against malicious software or malicious developers. Many of the security challenges for avionics systems can be addressed through the application of high assurance separation kernels.

2.2 The Principle of Least Privilege

In 1975, Saltzer and Schroeder concluded that *least privilege* is one of the eight design principles that can reduce design flaws [39]. They defined least privilege by stating "every program and every user of the system should operate using the least set of privileges necessary to complete the job. Primarily, this principle limits the damage that can result from an accident or error. It also reduces the number of potential interactions among privileged programs to the minimum for correct operation, so that unintentional, unwanted, or improper uses of privilege are less likely to occur." A decade later, the U.S. Department of Defense included a similar definition of least privilege in the Trusted Computer System Evaluation Criteria (TCSEC) [43]. Layering, modularity and information hiding are constructive techniques for least privilege that can be applied to the internal architecture of the underlying trusted foundation (e.g., separation kernel) to improve the system's resistance to penetration. The kernel can also be configured to utilize protection mechanisms such as access control and fine-grained execution domains to limit the abilities of a subject so that it is constrained to perform only the tasks for which it is authorized.

An example of the enforcement of least privilege is in modern operating systems such as would execute on a VMM. These operating systems require a minimum of two *protection modes*: one for the OS itself and one for application processes executing under the control of the OS. The operating system will view its relationship to applications as privileged. It will have access to virtualized privileged instructions that will permit it to manage memory, set and reset protection bits, control shared I/O resources, etc. Application processes are supposed to be less privileged. Without a mechanism such as multiple protection modes to enforce least privilege, a typical operating system executing within a VMM block would be at the mercy of its application processes, just as the VMM or separation kernel would be at the mercy of the operating system without similar hierarchical separation. By including a least privilege mechanism, presumably by utilizing hardware support, it is possible to provide both the separation kernel and the operating system with a simple, high assurance self-protection mechanism.

2.3 High Assurance and Least Privilege

The Trusted Computer System Evaluation Criteria (TCSEC) specifies the principle of least privilege as a system architecture requirement for high assurance evaluation classes (B2 to A1) [43]. While this requirement is noticeably absent in version 2.1 of the CC, even at the highest evaluation assurance level (EAL7), efforts are underway to correct this omission[31]. Despite the lack of an explicit reference to the principle of least privilege, the Common Criteria (CC) provides the groundwork for it in several ways. It defines assurance as "grounds for confidence that an entity meets its security objectives." [4] The CC explains that the correctness and effectiveness of the security functions are the primary factors for establishing the assurance that security objectives are met. A high assurance separation kernel must be proven to correctly implement the security functions defined in its specifications and effectively mitigate risks to a level commensurate with the value of the assets it protects. To complement the formal proof, a constructive analysis is used to demonstrate that the implementation maps to the specification. Thus, a focus on resource separation and the structured allotment of privileges affords simplicity to the separation kernel, and enables a high assurance analysis of the correctness of its implementation.

In practice, a separation kernel providing strict isolation is of little value. Controlled relaxation of strict separation allows applications to interact in useful ways, including participation in the enforcement of application-level policies. In the latter case, applications hosted on a separation kernel will need to be examined and evaluated to ensure that the overall system security policies are enforced. A monolithic application that runs with the same set of privileges throughout all of its modules and processes is hard to evaluate. In order to reason about the assurance properties of the system, the applications should be decomposed into components requiring varying levels of privilege. Such decomposition is more meaningful if the privilege boundaries are enforced by the separation kernel, rather than relying on, for example, error-prone ad hoc agreements between programmers or integrators. The principle of least privilege affords a greater degree of scrutiny to the evaluation of both the kernel and the application, resulting in a higher level of assurance that the overall system security objectives are met.

3 Inter-Block Flows

The first-order goal of a separation kernel is to provide *absolute* separation of the (effects of) activities occurring in different blocks. In practice, however, separation kernels are often used to share hardware among kindred activities that have reason to communicate in some controllable fashion. If there were not some such commonality, the need to share hardware would be less compelling. In other words, absolute separation of blocks is not so useful. Therefore, we include in the separation kernel a policy and mechanism for the controlled sharing of information between blocks.

The control of information flow between blocks can be expressed abstractly in an access matrix, as shown in the example of Table 1. This allows arbitrary sharing to be defined, establishing the inter-block flow policy to be enforced on the separation kernel applications.

	Block A	**Block B**	**Block C**
Block A	RWX	W	-
Block B	-	RWX	W
Block C	-	-	RWX

Table 1. Example Block-to-Block Flow Matrix

Notice that an inter-block flow policy in which the flow relationships partially order the blocks, such as in Table 1, may be suitable for the enforcement by the separation kernel of a multilevel confidentiality or integrity policy if meaningful sensitivity labels are immutable attributes of the blocks.[1] In this example, information flows (represented by ⇒) form the following ordering: Block A ⇒ Block B ⇒ Block C. An assignment of labels to these blocks in conjunction with the rules defined in Table 1 results in a recognizable multilevel security policy:

[1] Because a static separation kernel does not change the policy or resource allocation during execution, and assuming that the policy is not changed while the separation kernel is shut down, the policy may be considered to be global and persistent, viz. non-discretionary.

Block A := Unclassified
Block B := Secret
Block C := Top Secret

The block-to-block flow policy allows all of the information in a “source” block (e.g., Block A, above) to flow to every element of a “target” block (e.g., Block B, above). Extending the Table 1 scenario, if block B is also allowed to write to block A, for example to implement a downgrade function with respect to the assigned labels, then all of the code or program(s) in block B would need to be examined to ensure that their activities correspond to the intended downgrading semantics. If this assurance of correct behavior cannot be provided, such a circular flow (A ⇒ B ⇒ A) would create, in effect, one large policy equivalence class consisting of all of the information in blocks A and B.

To limit the effects of block-to-block flows, we next introduce the notion of controlling how much information is to be allowed to flow between and within blocks.

4 Least Privilege Flow Control

The implementation of a separation kernel results in the creation of active entities (subjects) that execute under the control of the separation kernel and the virtualization of system resources exported at the kernel interface (see Figure 1). Historically, many security models have utilized the abstraction of an *object*.[25] Objects have been classified in various ways (such as data object, storage object and named object) and there have been various approaches for defining devices and other system resources in terms of objects. We decided to avoid this nomenclature issue by simply modeling "resources." Similarly, as the definition of “resources” includes the abstractions that are exported by the separation kernel, "subjects" are defined to be a type of resource.

Resources are defined as the totality of all hardware, firmware and software and data that are executed, utilized, created, protected or exported by the separation kernel. *Exported resources* are those resources (including subjects) to which an explicit reference is possible via the separation kernel interface. That interface may include programming, administrative, and other interfaces. In contrast, *internal resources* are those resources for which no explicit reference is possible via the separation kernel interface.

Certain implementations of separation kernels elect to describe the system only in terms of blocks and have not chosen to describe the active system entities that cause information flow. Since the concept of subjects is a term of art [25], and for good reason, we will use it to describe the active entities in the separation kernel. We have found the use of the subject abstraction to be indispensable for reasoning about security in secure systems. Without the subject abstraction, it may be difficult to understand, for example, which block in a partitioned system is the cause of a flow between blocks (e.g., the flow could have been caused by the receiving block as a reader or by the sending block as a writer), which application programs within a block need to be trusted (e.g., evaluated with respect to the security policy), and how to minimally configure the programs and resources of such a system to achieve the principle of least privilege. Just as when writing prose, if actions are described *passively* (i.e., not attributable to the subject of a sentence) the cause of the action can be ambiguous. In addition, use of subjects permits construction of a resource-to-block allocation that provides a minimal configuration for least privilege (see Section 4.3).

Modeling of subjects within a partition also allows the representation and examination of more complex architectures such as multiple rings of execution, as well as multithreaded and multi-process approaches.

The following figure shows an example separation kernel system with three blocks, three subjects, a set of other resources, and some designated flows.

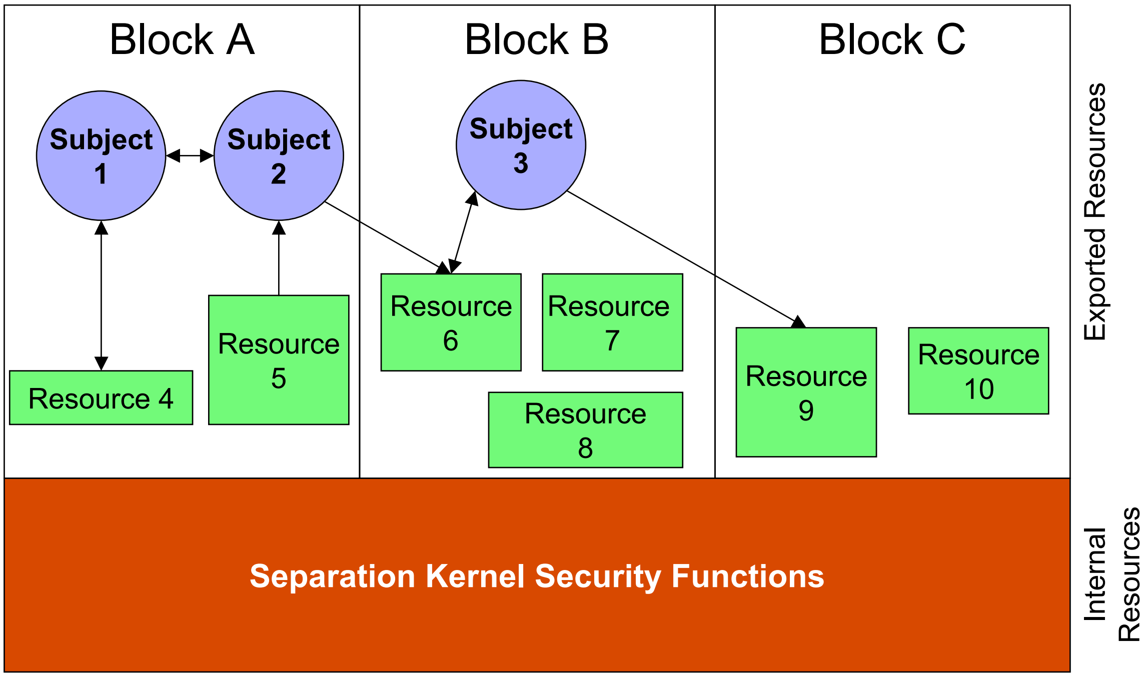

Figure 1. Example Separation Kernel Configuration

4.1 Subject-to-Block Flow Control

Thus far, blocks have been presented as an abstract representation of the separation policy. In the implementation of that policy, the separation kernel will allocate subjects and other exported resources to blocks. Table 2 illustrates such an allocation, i.e., a "tagging" of each subject and resource with its partition (per Figure 1). Of the resources described in this table, the first three are subjects and the remaining exported resources are passive. Each block must have at least one resource allocated to it since an empty block is useless and invalid. It is worth noting that an active entity need not be allocated to a block, and some blocks might contain only a subject, however the latter might be considered to be rather hypothetical and unlikely in concrete implementations. Every resource is allocated to one and only one block. Consequently, we can state that the blocks of the separation kernel constitute a partition (in the mathematical sense) where: R is the nonempty set of resources r and B is a nonempty set of subsets of R such that each element of R belongs to exactly one of the elements of B. From elementary set theory, it is known that a partition, B, can be used to create an equivalence relation on R. Thus we may induce that the allocation of resources to partitions creates equivalence classes.

Resource ID	Block
1	A
2	A
3	B
4	A

5	A
6	B
7	B
8	B
9	C
10	C

Table 2. Resource to Block Allocation

Having provided a means of distinguishing different subjects within a block, we can formulate a flow policy in which different subjects of a block are allowed to have different flows, to various blocks. This allows the flow policy from Figure 1 to be expressed in terms of subjects and blocks as illustrated in the access matrix of Table 3.

Subjects	Block A	Block B	Block C
1	RWX	-	-
2	RWX	W	-
3	-	RWX	W

Table 3. Subject-to-Block Flow Rules

The application domain attributes used to determine the allocation of resources to blocks, together with the flow rules, define a specific policy. Examples include non-discretionary policies that restrict information flow based upon confidentiality or integrity requirements. Non-security attributes could also be used as the basis for the partitioning. For example, if CPU time is described as a set of time units, the scheduling policy could be used to define blocks, with some blocks allocated more time units than others.

Within a block, it may be necessary to perform certain transformations on information that change its security attributes. For example, encryption may be viewed as a transformation of information from sensitive to unsensitive; a guard [2] performs information review and downgrading functions; and a quality assurance manager transforms prototype code into production code by regarding it in terms of reliability. For each of these transformation and transactional examples there may be several subjects within a block performing various aspects of the task at hand. The principle of least privilege requires that each of these subjects be given only the privileges required to do its particular task and no more. The separation kernel can support this objective by assigning access rights appropriately to the subjects within the block.

4.2 Subject-to-Resource Flow Control

Just as we provided subjects within a partition with different flow rules, we can also define separate rules for accessing different resources within a block. Table 4 illustrates how allocations to support the principle of least privilege are possible when the separation kernel supports per-subject and per-resource flow-control granularity: no subject is given more access than what is required to allow the desired flows.

	Resource ID									
Subject	1	2	3	4	5	6	7	8	9	10
1	-	RW	-	RW	-	-	-	-	-	-
2	RW	-	-	-	R	W	-	-	-	-
3	-	-	-	-	-	RW	-	-	W	-

Table 4. Subject-to-Resource Flow Matrix

Together, Tables 2 and 4 show abstract structures required to allow only the flows illustrated in Figure 1. It is clear from that the corresponding Block-to-Block flow matrix in Table 1, by itself, would allow many more flows than those illustrated in Figure 1.

An example of a useful application scenario for least privilege separation is that of a downgrader, shown in Figure 2. Block D would be unclassified, Blocks A, B and C would be classified, and Block C would hold the downgrader program. An untrusted initiator (UInit) process in A would write selected classified information to a classified holder buffer in Block A. An untrusted copier process would move the contents of the holder to the dirty search workspace in Block B. An untrusted dirty-word search process (UDWS) in B would provide advisory confirmation that the information was “suitable” for downgrading and copy the information into the clean results buffer (note that this process’s actions would be considered “advisory” with respect to the multilevel policy, rather than “trusted,” since it has no capability to actually violate the policy, nor to guarantee that the information is actually suitable for downgrading). Then the trusted downgrader (TDG) program in C would read the information from the results buffer and write it to an unclassified receiver buffer in D where it may be accessed by an untrusted end-point process (UEnd). As constrained by least privilege provided by the subject-to-resource flow matrix, the downgrader process in Block C would not be able to read from any resource other than the clean results and not be able to write to any resource in D other than the receiver. This would limit damage in the event of errors, for example in the downgrader, initiator or search processes, and contributes to a substantive argument that only the downgrader program need be *trusted* with respect to the application-level multilevel policy (viz., *depended* on to write down only when appropriate), and thus requiring of security verification with respect to that policy.

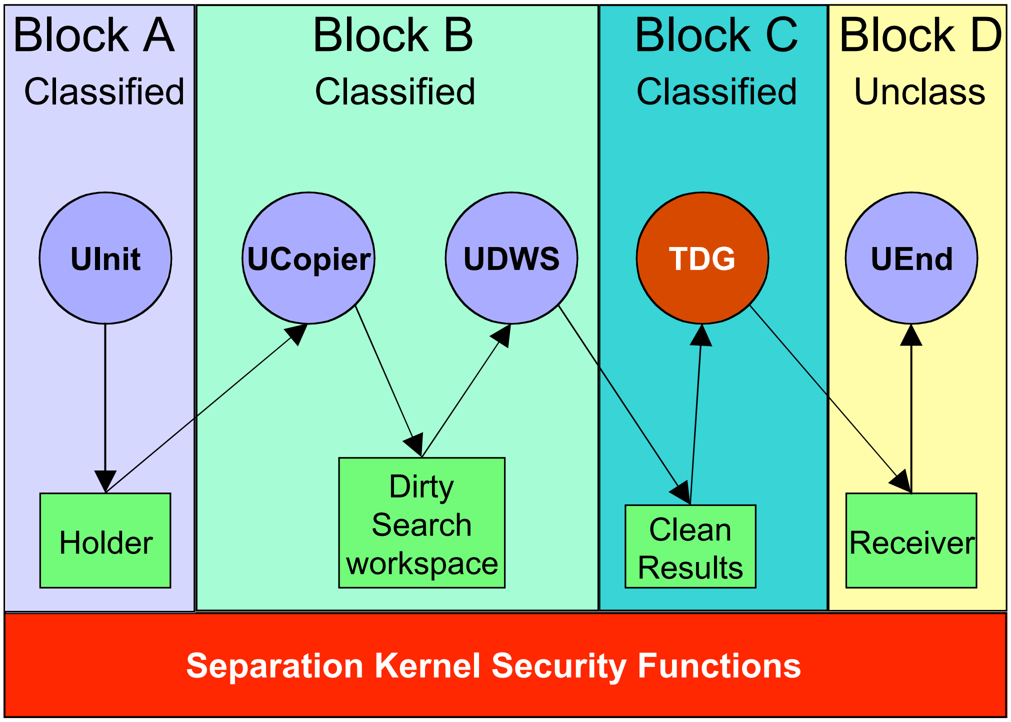

Figure 2. Trusted Downgrader. Dark areas with white text are trusted.

4.3 Alternative to Enforcement Of Least Privilege at Subject/Resource Level

An alternate conceptual approach to achieving the principle of least privilege for a system that enforces a Block-to-Block policy without a Subject-to-Resource matrix is through the use of a minimal configuration. A minimal configuration of a system is where, for all pairs of blocks, X and Y, all of the subjects of block X require for their functionality, and are given, the same access to all of the resources of block Y. Note that X and Y may be the same or different blocks, and the access given may be empty for a given pair of blocks. By definition, then, the Block-to-Block matrix for a system in a minimal configuration does not provide any more access than what is required. However, it is clear that minimal configurations are far less expressive than the more general configuration utilizing the Subject-to-Resource matrix, and would not be suitable for many types of secure systems, such as those that support execute-only memory.

5 Least Privilege Separation Model

This section provides a formal presentation and discussion of the least privilege separation model that supports two orthogonal flow policies: 1) block-to-block flow control and 2) least privilege flow control between subjects and resources.

The system elements of a least privilege separation model are: a set of resources `R`, a set of operations `O`, a set of modes of flow `F`, a *partitioning* of resources into a set of blocks `B`, an operation-to-effects function `MM`, a block-to-block flow function `BB`, and a subject-to-resource flow function `SR`:

```
  system = (R, O, F, B, MM, BB, SR)
```

Given the set of resources R and the partition of those resources B, we derive the function `RB`:

$$\mathrm{RB} : \mathrm{R} \rightarrow \mathrm{B}$$

such that $\mathrm{RB}(r) = b$ if and only if resource $r \in \mathrm{R}$ is an element of block $b \in \mathrm{B}$. Since B is a partition of R each element of R is in exactly one element of B.

We also provide a rule for interpreting the elements to define a secure system (see below). The model, representing a system with static resource assignments and static security policy, does not include any changes to the system elements: all functions are constant.

The set of resources, R, represents the totality of all hardware, firmware, software and data that are executed, utilized, created, protected or exported by the separation kernel. Resources are virtualized by the separation kernel such that each block's resources appear to be entirely its own. For example, memory may be virtualized into address spaces and/or objects that are assigned to blocks. The CPU may be virtualized into scheduling units that are assigned to individual blocks (e.g., cpu_unit(1..n)). For example:

$$\mathrm{RB}(\mathrm{cpu\_unit}.1) = b$$

Resources are divided into internal resources R_i, which are reserved for the kernel's own use and exported resources R_e , which are accessible through a defined interface to the kernel:

$$\mathrm{R}_i \subset \mathrm{R}$$
$$\mathrm{R}_e \subset \mathrm{R}$$
$$\mathrm{R}_i \cap \mathrm{R}_e = \emptyset$$

Subjects are abstractions created and controlled by the separation kernel that represent the active entities of the system (e.g., program, process, agent). Each subject $r_s \in \mathrm{R}_s$ is a resource from R_e, thus:

$$\mathrm{R}_s \subset \mathrm{R}_e$$

Each mode of flow f is selected from the set F. The mode indicates the direction of information flow between two entities, such as between a subject and a resource. F includes read and write, where semantically, write is a one directional flow from the subject to the resource that does not include the inverse flow (read). The modeling of an execute mode will depend on the system implementation, for example whether the hardware allows an *execute_only* mode of access, and whether the resulting flow (if any) is different from read with respect to the policy.

The set of all possible flow effects E is constructed from $\mathrm{R}_S \times \mathrm{R} \times \mathrm{F}$. A flow effect, e, defines a flow between a subject and a resource (direct flows between non-active resources are not allowed):

$$e = (r_s, r, f)$$

Each operation, $o \in \mathrm{O}$, available to subjects is associated by function MM with a set of flow effects, which may be the empty set, from the power set, $\wp$, of E:

$$\mathrm{MM}: \mathrm{O} \rightarrow \wp(\mathrm{E})$$

The function MM represents all the flows between pairs of resources that will be actualized by system operations.

The function SR defines the set of allowed flow relations between subjects and external resources, the least privilege flow policy:

$$\mathrm{SR}: \mathrm{R}_s \times \mathrm{R}_e \rightarrow \wp(\mathrm{F})$$

The function BB defines the set of allowed flow relations between blocks, called a BB_flowset, which is the system block-to-block flow policy:

$$\mathrm{BB}: \mathrm{B} \times \mathrm{B} \rightarrow \wp(\mathrm{F})$$

The functions BB and SR can be represented as matrices with subsets of F as the elements of the matrix, and are so illustrated elsewhere in this document.

From the set of relationships defined by BB, we can determine in which direction the information may flow between blocks. The shorthand FLOWS(BB,a,b) means that information may flow from block a to block b, implying that either BB(a, b) = write, or BB(b, a) = read must be true, or there exists a block c such that FLOWS(BB,a,c) and FLOWS(BB,c,b). FLOWS(BB,a,a) is implicit for all a ∈ B.

If information is allowed to flow circularly in a set of blocks, e.g. in both directions between two blocks, then a logical equivalence class of information results, since all information can be shared between the blocks. While the subject-to-resource flow controls can be used to prevent inter-block flows otherwise allowed by the block-to-block flow rules, it is generally intractable to determine which information in a block will be (e.g., transitively) allowed to flow into another block once the flow is allowed by a block-to-block flow rule and a subject-to-resource flow rule. Thus, to provide a meaningful partitioning of resources, we would like the inter-block flows defined by BB to be partially ordered (PO):

```
PO(B,BB) ⇔
    ∀i,j,k ∈ B(
          FLOWS(BB,i,i) (true by definition)
          &
          FLOWS(BB,i,j) & FLOWS(BB,j,i) -> i = j
          &
          FLOWS(BB,i,j) & FLOWS(BB,j,k)
                -> FLOWS(BB,i,k) (true by definition))
```

In some cases, flows between blocks in contradiction of the partial ordering are useful when constructing an application (e.g., to support a guard or downgrader application), and may be considered secure if it can be assured that only the desired information can flow as intended. Therefore, if such an inter-block flow were allowed by BB, we would require that any subject (e.g., application) allowed by SR to utilize the flow must have a level of assurance that is adequate to protect the information in both the source and the destination blocks. We refer to such a subject as a *trusted subject*. Trusted subjects are a subset of the set of all subjects:

$$R_t \subset R_s$$

Given a partial ordering relation for B, Bbase, a *trusted partial ordering* for the system is defined as follows, where Bcontra is a subset of BB that potentially contains flows in contradiction to those identified in Bbase:

```
TPO(system, Bbase) ⇔
    ∃ Bcontra: BB_flowset (
          BB = Bbase ∪ Bcontra
          &
          PO(B,Bbase)
          &
          ∀ $r_s$, r ∈ R; f ∈ F(
                (f ∈ Bcontra(RB($r_s$), RB(r))
                &
                f ∈ SR($r_s$,r)
                &
                ~PO(B,(Bbase ∪ ((RB($r_s$), RB(r)) = f)))
                )
                      ⇒ $r_s$ ∈ $R_t$
    )     )
```

A system is secure with respect to the least privilege separation model if and only if all of the flows implemented by system operations meet the block-to-block and least privilege flow policies and the system provides a trusted partial ordering with respect to an identified partial ordering of B:

```
SECURE(system, Bbase) ⇔
    ∀ e ∈ E, o ∈ O (
          e ∈ MM(o)
            ⇒  e.f ∈ BB(RB(e.$r_s$), RB(e.r))              (1)
                &
                e.r ∈ $R_i$ | e.f ∈ SR(e.$r_s$, e.r)          (2)
    )
    &
    TPO(system, Bbase)
```

5.1 Discussion of the Model

The intention is that SR represents a subset of the flows of BB, to provide an additional "least privilege" restriction to BB, but validity of the security predicate does not depend on this subset relationship. The formulation of SR to reflect the least amount of privilege required for a subject to perform its functions is assumed to occur during installation of the system's applications, by personnel familiar with each specific application. Since the applications will be fixed in a given static embedded system, this appears to be a reasonable expectation. In more dynamic systems, more algorithmic approaches [42] to determining and/or assigning "least" privileges might be in order.

Note that flow effects are specified in terms of the flows between all possible subjects R_S and resources R. Since R includes both *internal* and *exported* resources, all possible system flows are represented, and therefore the desired block separation property is represented, as well as the control of flow between blocks and the ability to enforce least privilege. If BB is empty, then the system is configured with a strict separation policy, viz., allowing no flows between blocks. If E were taken to represent only exported resources, then this model would need to assume the separation property regarding internal resources, either implicitly, axiomatically, or through the inclusion of additional properties [38][26].

One approach to formal system verification with this model would be to let E represent only exported resources, map the formal system specification to the model, and let a comprehensive covert channel analysis of the system and specifications provide the evidence for separation of internal resources. An alternative would be to let E represent both internal and exported resources, and provide a formal system specification that is at the same level of detail and abstraction as the system implementation – enabling the representation of all internal and external flows. The latter alternative, to be practical for a system of useful size, would seem to require either the ability to automatically generate a detailed specification from an abstract model, or automatically generate the code from the detailed specification, so that the system would not need to be implemented redundantly in separate logics (viz., specification and code). However, providing such detail in the specification might defeat one of its primary purposes, which is to provide an understandable representation of the design.

The characterization shown here of the relationships of the significant internal elements of a static separation system results in a simple, yet complete model of the desired security policy. Since the system in execution will not allow changes to "flows" granted or implemented, constant functions are adequate for representing the security elements of the system. Some might ask whether this model is "too simple" to be interesting, to which we would reply that the ability to clearly represent the security policy of a significant class of processing engines appears to be useful. As a precursor to further development, we have represented this model in the Ina Jo specification language [40], and proven that information flows resulting from representative system calls are bound by the security predicate (Ina Jo was chosen for its built-in support for inter-level mapping, which will be utilized when proving that the formal system specification conforms to the formal model).

It could be argued that some of the traditional separation kernel models could allow a granular definition of blocks, thereby providing "least privilege;" for example if each subject and resource were assigned to their own block. However, such a definition would eliminate the ability of those "one-dimensional" models (recall that our model presents two orthogonal sub-policies) to portray a higher-level policy regarding flows between equivalence classes of subjects and resources, such as is useful in dealing with multilevel security at the application level.

In an ideal system, no covert timing or storage channels will be present. In an implementation, covert channels between blocks would be visible as a violation to (1) for a particular `e`, where `e.r` is an internal resource. One sort of timing channel might show up as the modification to the length of an internal structure by one subject, which is indirectly visible to a subject in another block. Since the channel is via an internal resource, it can only be accessed indirectly and not directly via a kernel "handle" or name of some sort. The observing subject would measure the difference in the length of the structure through observing the difference in its own execution time for an operation that accessed that structure. Similarly, exhaustion of the same internal resource might result in an error message, and corresponding storage channel.

A modular foundational component such as a separation kernel may be intended to support systems that provide and enforce more general and flexible policies. To minimize the expense of building and evaluating the larger systems, it would be ideal to be able to use the formal model and other verification evidence of the separation kernel, without re-verification of the kernel itself. One can then anticipate the need to formally compose the separation kernel model with the general-purpose model of the larger system. Additional research is needed to investigate approaches for the composition of the principle of least privilege separation model with other formal models.

5.2 Implementation Considerations

Inter-process communications can be implemented in a system as direct subject-to-subject operations, or indirectly via different sorts of communication and synchronization resources (e.g., synchronization objects [35]). The Subject-to-Resource matrix will allow either of these types of communications to be modeled.

The class of separation kernels we are concerned with utilizes memory-mapped I/O, such that input and output between programs and devices occurs via memory segments that are in the address space of the program. Therefore, whether devices are in the kernel, or are themselves active entities (subjects) in the user domain, they need not be represented explicitly in the model.

6 Related Work

6.1 Separation Kernel Formal Models

Rushby [38] provided the seminal effort on the formal modeling of separation kernels. This work modeled the separation between blocks (referred to as "regimes") of a partitioned system, such that the active entity(s) of a block cannot perceive, by observing their own input and output, whether or not there are any other blocks present on the system. The Rushby model does not represent the control of flows between blocks. In contrast, the work presented here models the security of a system in terms of the separation and control of allowed inter-block flows, as well as the control of flows to achieve least privilege.

Hartman [19] extended the Rushby model to represent the control of communication between blocks (where each "process" of the model represented its own block). The model was developed for a separation kernel to run on the SCOMP processor, which featured four hardware-privilege levels of execution within each process [16]. Communication between blocks is modeled as occurring via memory buffers. Although the SCOMP hardware provided four privilege levels, control of flows within a block (e.g., applied to threads or subjects assigned to different privilege levels of a SCOMP process) is not represented, nor is flow control with respect to resources other than memory buffers, so the "least privilege" control we describe is not representable using this model.

The MASK formal model, briefly described by Martin [26], characterizes the separation of blocks (referred to as "cells"), as well as the control of message-based communication between blocks, but does not represent the ability to provide least privilege with respect to resources or activities within a block (cell).

Alves-Foss [1] provides a formal model for multilevel security enforcement by message-passing applications, such as multilevel secure network interface units, that are to be built on a separation kernel. This model assumes the effective separation between blocks, and provides a framework specifying the secure behavior of applications, which utilize allowed flows between blocks. It does not describe the granularity of control required to represent the principle of least privilege.

Schellhorn, et al [41] describe a formal model that includes non-interference between blocks (called "applications"), as well as a mandatory access control (MAC) policy regarding both confidentiality and integrity for references to resources within a block. There is one subject per block, which is associated with a set of private resources ("files"). A "channel" mechanism is defined to allow inter-block communication at the granularity of blocks, but to no greater granularity, so the principle of least privilege is not applicable to inter-block accesses. The MAC mechanism allows least privilege to be enforced on a subject's access to resources within its own block, but least privilege restrictions are not expressible for active entities at a finer granularity than the entire application/subject/block.

Recently, Greve, et al [18] provided a model for a separation kernel accompanied by a description of several derived properties and the abstract specification for a secure guard application based on the model. This model represents flow control at the granularity of memory segments within a block (referred to as a "partition"), which has the potential to represent the principle of least privilege with respect to resources within a block. This model differs from ours, however, in that it does not include logic for expressing a higher-level policy regarding flows between blocks.

6.2 Protection Profiles for Separation Kernels

The Common Criteria security evaluation paradigm includes a document called a *protection profile* that specifies the security functionality and assurance for an entire class of IT products, as well as a document called a *security target*, which provides a similar specification for a specific IT product. The protection profile is evaluated for consistency with the Common Criteria requirements for protection profiles; the security target is evaluated for consistency with the Common Criteria requirements for security targets, as well as for consistency with an identified protection profile (if any); and finally the product is evaluated against the security target. [4]

A protection profile for high assurance "partitioning kernels" was initiated under the auspices of The Open Group by a team directed by the Lockheed Martin Aeronautics Company and the Air Force Research Laboratory. A draft of this protection profile was produced in March of 2002 [31] for the National Security Agency. This protection profile does not address the principle of least privilege.

To complement this effort we are part of a group writing a high robustness protection profile for separation kernels under the Common Criteria [31]. The proposed inclusion of a least privilege requirement in this protection profile is being reviewed by the group.

6.3 Trusted Computing Exemplar Project

Separation kernel technology is being applied in our *Trusted Computing Exemplar* project[21]. This ongoing effort is intended to produce a high assurance least privilege separation kernel[31].

The primary security function of our Separation Kernel is to enforce subject and block separation, while providing primitive operating system services sufficient to support simple applications. The *embedded* focus of the kernel drives several high-level design characteristics. It will have a static runtime resource configuration and its security policy regarding access to resources will be based on static process/resource access bindings, which are subject to offline configuration (e.g., via an access matrix, such as are shown in Figures 1, 2 and 4). The kernel will create and support a fixed number of application-level processes established by the system configuration. Application processes will be scheduled in static fashion with each process being given a predetermined amount of time, set by the configuration. The static nature of resource allotment will provide predictable processing behavior, as well as limit the *covert channels* based on shared resource utilization [25][23][30]. Simple process synchronization primitives will also be provided, that can be implemented to be demonstrably free of covert channels [35].

6.4 Separation Kernels for Virtual Machine Monitors

By providing virtualization of all resources, including the hardware, a VMM [17][33][22]provides a rigorous form of separation and can be considered a subset of the general class of separation kernels. There are a number of VMM efforts and products where notions from separation kernel technology are potentially applicable. Both Connectix [8] and VMware [10] virtualize popular PC processors. VMware is a component of the NetTop prototype [28] being developed by the government. The ultimate objective of the NetTop effort is to provide virtual machines at different classification levels on a single commodity PC. The Xen project is constructing a highly efficient, open source virtual machine monitor [3]. A specialized system with an embedded virtual machine is the planned for the Software Defined Radio [14]. It will take advantage of the Advanced INFOSEC Machine VLSI chip to support radio-based security including, key management, multi-algorithm management, multiple channel communications, isolation of plain text from cipher text, and separation of internal sensitive processing from external unclassified processing. The target applications for this technology are the U.S. Government Digital Modular Radio and the Joint Tactical Radio System [24][7].

Both the Terra architecture [15] and Microsoft's NGSCB (previously called Palladium) [5][11][29][12] present a specialized virtual machine monitor that will partition the system into exactly two domains, trusted and untrusted, where different operating systems will be executed. In both cases, commodity software can be executed in the open or untrusted virtual machines. The trusted/closed VMs are isolated so that neither the platform owner nor malicious software is able to observe or modify their contents. For these VMMs, the principle of least privilege is essential so that the operating systems executing in the VMs can be protected from the applications and the VMM itself can be protected from both.

7 Conclusion

The separation kernel abstraction and the principle of least privilege are significant tools for the protection of critical system resources. Separation kernels are increasingly being used as a foundation in systems to provide such protection. It is important that a model suitable for high assurance is available for the evaluation of these systems. In this paper, we presented the least privilege separation model, which is the fusion of the separation abstraction and the least privilege principle.

In addition to the inter-block flow control policy prescribed by the traditional separation kernel paradigm, our proposed approach supports an orthogonal finer-grained flow control policy by extending the granularity of protected elements to subjects and resources, as well as blocks, in a partitioned system. We showed how least privilege applied to the actions of external subjects provides enhanced protection for secure systems, and also discussed how subject abstraction is a critical tool for information flow and security analysis.

We examined the relationship of allowed inter-block flows to resulting policies enforced on separation kernel applications. An interesting observation is that if the flows between blocks are partially ordered, then the separation kernel can be seen to enforce a multilevel-secure mandatory access control policy (assuming that the partial ordering is maintained in a global and persistent manner). We have introduced the notion of a trusted partial ordering to describe flows between blocks that are contrary to the multilevel policy. Another observation is that circular flows between blocks do not result in a partial ordering, unless all of the blocks involved in the circularity are collapsed into a single equivalence class (meta-block).

In summary, application of the principle of least privilege, resource separation and controlled sharing are synergistic security properties in a separation kernel. Each subject is only given a minimum set of logically separated resources necessary to perform its assigned task, and the sharing of resources between subjects is rigorously controlled by the kernel. A separation kernel that correctly implements these properties can meet the objective to minimize and confine damage with a high level of assurance.

Acknowledgements

We would like to thank George Dinolt for his helpful input.

References

[1] Alves-Foss, J., The architecture of Secure Systems, in *Proc. Hawai'i International Conference on System Sciences: Emerging Technologies Track*, January, 1998.

[2] Anderson, J.P., On the Feasibility of Connecting RECON to an External Network, Tech. Report, James P. Anderson Co., March 1981.

[3] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I, Warfield, A., Xen and the Art of Virtualization, In *Proceedings of the 20th ACM Symposium on Operating System Principles*, Bolton Landing, NY, October 2003, pp. 164-177.

[4] Common Criteria for Information Technology Security Evaluation, Part 1: Introduction and General Model, CCIMB-99-031, Version 2.1, August 1999.

[5] Carroll, A., Juarez, M., Polk, J., and Leininger. T., Microsoft Palladium: A business overview. http://www.microsoft.com/PressPass/features/2002/jul02/0724palladiumwp.asp, August 2002.

[6] Chincheck, S. J., Programmable Embeddable INFOSEC Product, http://www.nrl.navy.mil/content.php?P=03REVIE159

[7] Command, Control, Communications, Computers and Intelligence, Joint-Service/Navy-Wide Systems, http://www.chinfo.navy.mil/navpalib/policy/vision/vis02/vpp02-ch3u.html.

[8] Connectix, Product Overview – Datasheet, 2003, http://www.connectix.com/support/vpcw_online.html.

[9] Denning, D., A Lattice Model of Secure Information Flow, *Communications A.C.M.*, 19(5):236-243, 1976.

[10] Devine, S., Bugnion, E., and Rosenblum, M., Virtualization System Including a Virtual Machine Monitor for a Computer with a Segmented Architecture. US Patent, 6397242, October 1998.

[11] England, Paul and Peinado, Marcus, Authenticated operation of open computing devices. In *Proceeedings 7th Australasian Conference on Information Security and Privacy*, Springer-Verlag, Lecture Notes on Computer Science 2384, pages 346–361, August 2002.

[12] England, P., Lampson, B. , Manferdelli, J., Peinado, M. and Willman, B., A Trusted Open Platform, *IEEE Computer*, Vol. 36, No. 7, pp. 55-62, 2003.

[13] Gasser, M., *Building a Secure Computer System*, Van Nostrand Reinhold, New York, N.Y., 1987.

[14] General Dynamics Decision Systems, AIM – Advanced INFOSEC System: System Overview, 2003. Also AIM data sheet aim08162002.pdf.

[15] Garfinkel, T., Pfaff, B, Chow, J., Rosenblum, M., Boneh, D., Terra: a Virtual Machine-based Platform for Trusted Computing , In *Proceedings of the Nineteenth ACM symposium on Operating Systems Principles*, Bolton Landing, NY, October 2003, pp. 193 – 206.

[16] Gligor, V., Analysis of the Hardware Verification of the Honeywell SCOMP, In Proceedings of the 1985 IEEE Symposium on Security and Privacy, Oakland, CA, April, 1985, pp 32-43.

[17] Goldberg, R., Architectural Principles for Virtual Computer Systems, Ph.D. thesis, Harvard University, Cambridge, MA, 1972.

[18] Greve, D., Wilding, M., Vanfleet, W.M., A Separation Kernel Formal Security Policy, *Fourth International Workshop on the ACL2 Theorem Prover and Its Applications*, July 13-14, 2003, Boulder, Colorado.

[19] Hartman, B. A., A Gypsy-Based Kernel, In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May, 1984, pp. 219-225.

[20] Intel Architecture Software Developer's Manual, Vol. 3: System Programming Guide, Intel Corporation, Santa Clara, CA, 1997.

[21] Irvine, Cynthia E., Levin, Timothy E., Nguyen, Thuy D., and Dinolt, George W., The Trusted Computing Exemplar Project, Proceedings of the 2004 IEEE Systems, Man and Cybernetics Information Assurance Workshop, West Point, NY, June 2004, pp. 109-115.

[22] Karger, P.A., Zurko, M.E., Bonin, D.W., Mason, A.H., Kahn, C.E., A VMM Security Kernel for the VAX Architecture, In *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May, 1990, pp. 2-19.

[23] Kemmerer, R.A., A Practical Approach to Identifying Storage and Timing Channels, In *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, Oakland, CA, April 1982, pp. 66-73.

[24] Joint Tactical Radio System, JTRS+brochure.pdf, JTRS Joint Program Office, Arlington, VA 22209 http://jtrs.army.mil/sections/referencedocuments/fset_referencedocuments.html?referencedoc_brochure.

[25] Lampson, B., Protection, In *Proceedings 5th Princeton Conference on Information Sciences*, Princeton, NJ, 1971. Reprinted in Operating Systems Reviews, 8(1): 18-24, 1974.

[26] Martin, W., White, P., Taylor, F.S., and Goldberg, A., Formal Construction of the Mathematically Analyzed Separation Kernel, In *Proceedings of the 15th International Conference on Automated Software Engineering*, IEEE Computer Society Press, Grenoble, France, September 2000.

[27] Marsh, C., Real-Time Operating Systems and Hardware Support, *Avionics Magazine*, June 2002 www.aviationtoday.com/reports/avionics/previous/0503/0503real_time.htm.

[28] Meushaw, R., and Simard, D., NetTop:Commercial Technology in High Assurance Applications, *Tech Trend Notes*, 9(4), 2000.

[29] Microsoft, Microsoft Next-Generation Secure Computing Base: An Overview, http://www.microsoft.com/resources/ngscb/ngscb_overview.mspx, April 2003.

[30] J.K. Millen, Covert Channel Capacity, Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, CA, pp. 60-66, April 1987.

[31] National Security Agency, U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness, 1 July 2004, http://niap.nist.gov/pp/draft_pps/pp_draft_skpp_hr_v0.621.html

[32] Object Management Group, Preliminary Draft: Partitioning Kernel Protection Profile, prepared by Rockwell Collins and Boeing, Version 0.3, 23 October 2002, Google cached HTML version of http://www.omg.org/docs/security/02-11-07.doc.

[33] Popek, G., and Goldberg, R., Formal Requirements for Virtualizable 3rd Generation Architectures, *Communications of the A.C.M.*, 17(7):412-421, 1974.

[34] Preparata, F. P., and Yeh, R.T., Introduction to Discrete Structures for Computer Science and Engineering, Addison Wesley, Reading, MA, 1973.

[35] Reed, D.P., and Kanodia, R.K., Synchronization with Eventcounts and Sequencers, *Communications of the A.C.M.*, 22(2):115-123, 1979.

[36] RTCA, Software Considerations in Airborne Systems and Equipment Certification, Document No. RTCA/DO-178B, prepared by Special Committee 167 of RTCA, RTCA, Inc., Washington, DC., December 1992.

[37] John Rushby. Design And Verification Of Secure Systems, *ACM Operating Systems Review*, 15(5), 1981.

[38] John Rushby, Proof of separability: A verification technique for a class of security kernels, In *Proc. International Symposium on Programming*, Lecture Notes in Computer Science, 137:352–367, April 1982.

[39] Saltzer, J. H., and Schroeder, M. D., The Protection of Information in Operating Systems, *Proceedings of the IEEE*, 63(9):1278-1308, 1975.

[40] Scheid, J., and Holtsberg, S., Ina Jo Specification language Reference Manual, Paramax Systems Corporation, TM-6021/001/06, June 1992.

[41] Schellhorn, G., Reif, W., Schairer, W., Karger, P., Austel, V., and Toll, D., Verification of a Formal Security Model for Multiapplicative Smart Cards, In *Proceedings of the 6th European Symposium on Research in Computer Security (ESORICS 2000)*, Volume 1895 of Lecture Notes in Computer Science (LNCS), pages 17--36. Springer-Verlag Berlin Heidelberg, 2000

[42] Schneider, F. B., Least Privilege and More, *IEEE Security and Privacy*, IEEE Computer Society, 1(5):55-59, 2003.

[43] Department of Defense Trusted Computer System Evaluation Criteria, DoD 5200.28-STD, December 1985.

[44] Andrew Whitaker, et al., Scale and Performance in the Denali Isolation Kernel, In *Proceeding of the Fifth Symposium on Operating Systems Design and Implementation (OSDI 2002)*, December 2002.